Example 2-34. Enabling Back Face Culling with DirectX

To enable back face culling with DirectX, the following renderstate is set:

```
SetRenderState(D3DRENDERSTATE_CULLMODE, <MODE>);
```

MODE is D3DCULL_CCW for counter clockwise culling, or D3DCULL_CW for clockwise culling.

Example 2-35. Enabling Back Face Culling with OpenGL

To enable back face culling with OpenGL:

```
glEnable(GL_CULL_FACE);
glCullFace(<MODE>);
```

MODE is GL_FRONT, GL_BACK, or GL_FRONT_AND_BACK.

2.3 2D Capabilities

In this section the 2D capabilities of the Intel740™ graphics accelerator are discussed:

- “BitBLT Engine”(below)
- “Stretch BLT Engine” (below)
- “Color Expansion” (below)
- “Hardware Cursor” on page 2-44
- “Video Display Resolutions” on page 2-44

2.3.1 BitBLT Engine

The Intel740™ graphics accelerator’s high performance 64-bit BitBLT engine provides hardware acceleration for many common Windows operations. To facilitate these, there are two primary BitBLT functions in the Intel740™ graphics accelerator: fixed BitBLT and stretch BitBLT. Fixed BitBLT involves transferring blocks of data from one memory location to another. The capability of performing raster operations on the data using a pattern is also included. Stretch BitBLT can stretch source data in the X and Y directions to a destination larger or smaller than the source. Stretch BitBLT functionality expands a region of memory into a larger or smaller region using replication and decimation.

If required, the Intel740™ graphics accelerator will expand monochrome data into color data. This new data will be destination aligned. The main feature of the BitBLT is to take a stored pattern and expand it to the destination color space while destination aligning it.The Intel740™ graphics accelerator’s raster opcode engine supports all 256 Microsoft-defined raster operations (ROPs) including transparent BitBLT.

2.3.1.1 Fixed BitBLT

The rectangular block of data does not change as it is transferred between memory locations. The allowable memory transfers are between: AGP memory and local memory, local memory and AGP memory, AGP memory and AGP memory, and local memory and local memory. Data to be transferred can consist of regions of memory, patterns, or solid color fills. A pattern will always be 8 x 8 pixels wide and may be 1, 8, or 16 bits per pixel.

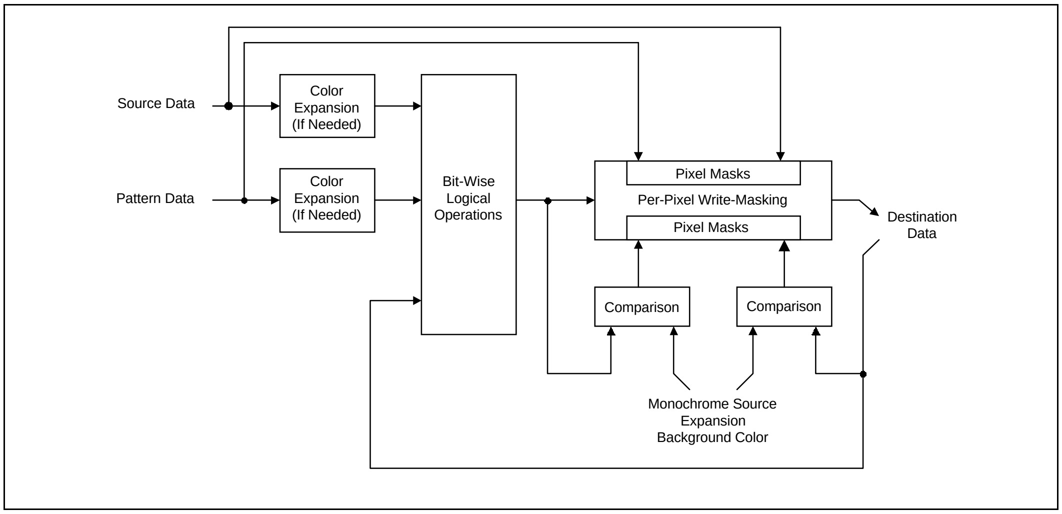
The Intel740™ graphics accelerator has the ability to expand monochrome data into a color depth of 8, 16, or 24 bits. BLTs can be either opaque or transparent. Opaque transfers, move the data specified to the destination. Transparent transfers, compare destination color to source color and write according to the mode of transparency selected.

Data is horizontally and vertically aligned at the destination. If the destination for the BLT overlaps with the source memory location, the Intel740™ graphics accelerator can specify which area in memory to begin the BLT transfer. Use of this BLT engine accelerates the Graphical User Interface (GUI) interface of Microsoft* Windows.

While the BitBLT engine is often used simply to copy a block of graphics data from the source to the destination, it also has the ability to perform more complex functions. As illustrated in Figure 2-21, the BitBLT engine can receive three different blocks of graphics data (source data, destination data, and pattern data). The source data can exist either in the Frame Buffer or it can be provided by the host CPU from some other source (e.g., AGP memory). The pattern data always represents an 8x8 block of pixels that must be located in the Frame Buffer, usually within the off-screen portion. The data already residing at the destination may also be used as an input, but this data must also be located in the Frame Buffer.

The BitBLT engine can use any combination of these three different blocks of graphics data as operands, in both bit-wise logical operations to generate the actual data to be written to the destination, and in per-pixel write-masking to control the writing of data to the destination. It is intended that the BitBLT engine will perform these bit-wise and per-pixel operations on color graphics data that is at the same color depth that the rest of the graphics system has been set. However, if either the source or pattern data is monochrome, the BitBLT engine has the ability to put either block of graphics data through a process called “color expansion” that converts monochrome graphics data to color. Since the destination is often a location in the on-screen portion of the Frame Buffer, it is assumed that any data already at the destination will be of the appropriate color depth.

Figure 2-21. BLT Engine Block Diagram and Data Paths



2.3.1.2 Stretch BLT Engine

The Stretch BLT Engine allows a source memory region to be blitted to a destination region which is larger, smaller or the same size as the source region by replacing or removing pixels. Expansion and shrinking can occur in both the horizontal and vertical directions.

An additional feature of the Stretch BLT Engine is the ability to transparently place the source data over some destination data by masking. This is useful for sprites in 3D games.

2.3.1.3 Color Expansion

During a BLT operation, source color depth may not be the same as destination color depth. The Intel740™ graphics accelerator supports monochrome data as well as 8, 16, and 24 bit color data. The BLT engine has the ability to expand source monochrome data into a color depth of 8, 16, or 24. Color expansion can be either opaque or transparent. When opaque, a foreground and background color are both transferred to the destination in the new color depth. When transparent, only the foreground color is specified. This is very useful for text data. Text data can be stored as one bit per pixel color (monochrome), and expanded to the correct color later.

2.3.2 Hardware Cursor

The Intel740™ graphics accelerator allows a total of 16 cursor patterns to be stored in 4 Kbytes. Six modes are provided for the cursor:

- 32x32 2 bpp 2-plane mode (Solid Color, Inverted Solid Color, Transparent, Inverted)
- 128x128 1 bpp 2-color mode
- 128x128 1 bpp 1-color and transparency mode
- 64x64 2 bpp 3-color and transparency mode
- 64x64 2 bpp 2-plane mode (Solid Color, Inverted Solid Color, Transparent, Inverted)
- 64x64 2 bpp 4-color mode

2.3.3 Video Display Resolutions

The Intel740™ graphics accelerator's video function provides analog output for use with a monitor or a 8/12-bit digital output to interface to a TV output chip. Integrated into the Intel740™ graphics accelerator is an I^2C interface to facilitate this capability. Video synchs and timings are fully programmable. Any overlays are merged with data from the frame buffer during output and can be scaled in the X and Y directions. Gamma correction can be applied on the video output. Resolutions supported for display ranges are shown in Table 2-8. In addition to the standard VGA modes, the Intel740™ graphics accelerator also supports the following extended modes with the stated memory and refresh timings:

Table 2-8. Display Modes Supported

Resolution	Bits Per Pixel (frequency: Hz)		
	8-bit Indexed	16-bit	24-bit
320x200	60,72,75,85	60,72,75,85	60,72,75,85
320x240	60,72,75,85	60,72,75,85	60,72,75,85
512x384	60,72,75,85	60,72,75,85	60,72,75,85
640x350	85	85	85
640x480	60,72,75,85	60,72,75,85	60,72,75,85
800x600	56,60,72,75,85	56,60,72,75,85	56,60,72,75,85
1024x768	60,70,75,85	60,70,75,85	60,70,75,85
1280x1024	60,72,75,85	60,72,75	—
1600x1200	60,75	—	—

The video display controller is responsible for the horizontal and vertical timings of the monitor, accessing data from memory, preparing data for display, and presenting the results to the monitor or TV. The Intel740™ graphics accelerator can convert YUV(4:2:2) to RGB format. An I^2C Bus is provided for easier connection to some chips.

The display engine also contains an overlay unit. The overlay (full motion video) unit is capable of converting from YUV4:2:2 format to 24 bpp RGB. Line widths to 720 pixels are supported. X,Y interpolation can be performed on the overlay window if the source is smaller or larger than the destination display size. The Intel740™ graphics accelerator performs filtering/smoothing when interpolating in the horizontal and vertical directions. The data may be scaled in both the horizontal or vertical direction using a six bit expansion value. On output, the data is scaled up. The image is increased in size only. This expansion is smoothed/filtered before being passed to the display.

When stretching is performed, the horizontal filter is 1-1. The vertical interpolation is either deblocking (average on change only) or 1-2-1 running average. Color keying is performed so that pixels of a selected color are transparent. (This editing effect is sometimes known as “blue screening.”)

The Intel740™ graphics accelerator contains a separate hardware cursor for Windows. The cursor information is not stored within the frame buffer but is combined with the screen image immediately before the image is displayed. Functionality built into the cursor allows it to be enabled or disabled. Up to 16 cursor patterns (depending on size) may be stored in separate cursor data space.

The combined result from the hardware cursor, overlay, and primary display is performed by the RAMDACs. There are three 8-bit DACs (one for controlling red, one for green, and one for blue). Each DAC has a 256x8 palette RAM is responsible for storing information about the colors to be displayed. The Intel740™ graphics accelerator is optimized for a 2D output resolution of 1024x768 and a 3D display resolution of 640x480. Within the 2D section, the horizontal sync, vertical sync, and blanking signals are fully programmable.

2.4 Video, VBI, and Intercast Capabilities

The Intel740™ graphics accelerator's Video, VBI, and Intercast capabilities are discussed in the following subsections:

- "Video Capture Port" (Section 2.4.1)
- "Video Overlay" (Section 2.4.2)
- "VBI and Intercast" (Section 2.4.3)

2.4.1 Video Capture Port

2.4.1.1 Overview

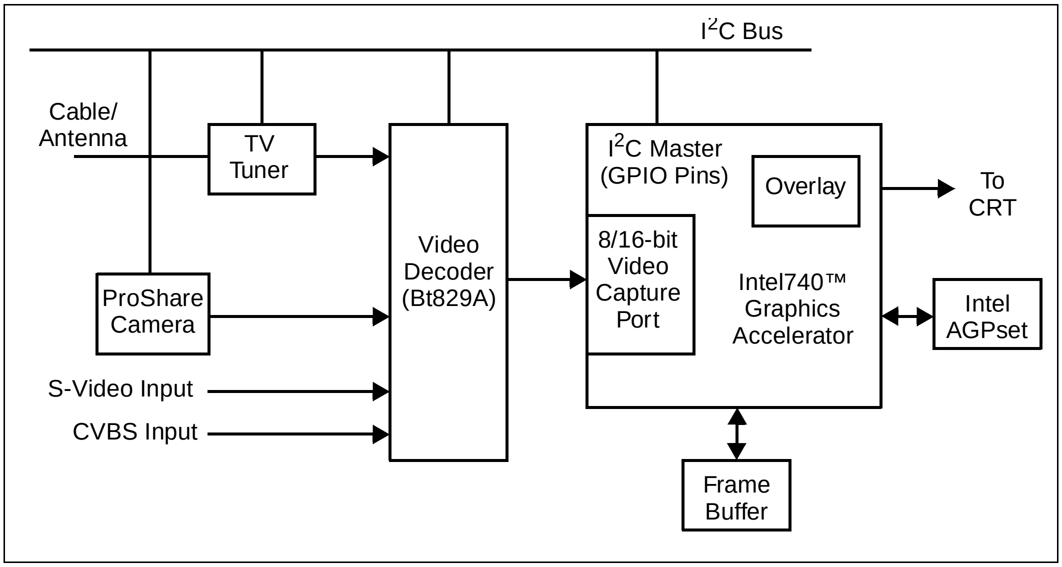
The PC video interface to the Intel740™ graphics accerlator is a unidirectional digital input port that accepts 16-bit wide data, two synchronizing signals (HREF, VFREF), and a pixel rate clock (VCLK). The video capture port can be configured as a VMI interface. Taking the digital video data from this video port, the Intel740 graphics accelerator can perform video functions such as color space conversion, scaling, zooming, interpolation, and video playback. Captured video is stored in a progressive format, as opposed to an interlaced format. See Section 2.4.2 for more information regarding the progressive format and video overlay. Although YUV 4:2:2 is the native format for this port, RGB-15, RGB-16, and RGB-24 input formats are also supported. Not all Intel740 graphics accelerator cards are configured to support video capture; be sure to refer to the card manufacturer's documentation to see if the card supports the video capture port.

Devices that output an analog signal can be connected to the video capture port through a third party chip that provides analog to digital conversion. Digital camera video conferencing applications are supported permitting the user to have an unflipped/mirrored view. This port provides support for Intercast technology and POTS (Plain Old Telephone Service) video conferencing. Note that an external third party VBI decoder chip is needed for Intercast technology. For POTS video conferencing, the port interfaces to a camera.

To facilitate digital camera applications, the Intel740 graphics chip can perform backward writes. This allows the user to see a mirrored or non-mirrored view on screen.

Gamma correction is also provided. When in 8-bit-per-pixel mode or smaller, the graphics data is expanded by a palette. If analog to digital conversion is needed, an external chip creates the digital signal sent to the Intel740 graphics chip.

Figure 2-22. Intel740™ Graphics Accelerator Video Capture System Diagram



2.4.1.2 Video Capture Programming

Video capture is supported through the Video for Windows (VfW) API. Below is an example of how to perform video capture using VfW with the Intel740 graphics chip. Refer to Microsoft* Video for Windows design kit documentation for further information regarding VfW capture.

Example 2-36. Capturing a Video Sequence Using the VfW API

Capturing a video sequence requires a series of parameters found in the CAPTUREPARMS struct to be initialized, followed by a call to capCaptureSequence(HWND). The following is a listing of the CAPTUREPARMS struct, followed by a brief example of how to capture a video sequence.

```
//***************************************************************************
//
// CAPTURE PARAMS struct definition
//
//***************************************************************************

typedef struct tagCaptureParms {
DWORDdwRequestMicroSecPerFrame;// Requested capture rate
BOOLfMakeUserHitOKToCapture;    // Show “Hit OK to cap”
                          // dialog
UINTwPercentDropForError;// Give error msg if > value
// default = 10%
BOOLfYield;        // Capture via background task?
DWORDdwIndexSize;  // Max index size in frames (32K)
UINTwChunkGranularity;// Junk chunk granularity (2K)
BOOLfUsingDOSMemory;// Use DOS buffers?
UINTwNumVideoRequested;// # video buffers, If 0,autocalc
BOOLfCaptureAudio;// Capture audio?
UINTwNumAudioRequested;// # audio buffers, If 0,autocalc
UINTvKeyAbort;     // Virtual key causing abort
BOOLfAbortLeftMouse;// Abort on left mouse?
BOOLfAbortRightMouse;// Abort on right mouse?
BOOLfLimitEnabled;// Use wTimeLimit?
```

```
UINTwTimeLimit;    // Seconds to capture
BOOLfMCIControl;   // Use MCI video source?
BOOLfStepMCIDevice;// Step MCI device?
DWORDdwMCIStartTime;// Time to start in MS
DWORDdwMCIStopTime;// Time to stop in MS
BOOLfStepCaptureAt2x;// Perform spatial averaging 2x
UINTwStepCaptureAverageFrames;// Temporal average n Frames
DWORDdwAudioBufferSize;// Size of audio bufs (0=default)
BOOLfDisableWriteCache;// Attempt to disable write cache
UINTAVStreamMaster;// Which stream controls length?
} CAPTUREPARMS;

//****************************************************************************
//
// Video Sequence Capture Example
//
//****************************************************************************

HWNDghWndCap;
extern CAPTUREPARMSgCapParms;
BOOLfResult;
charvidName[] = "C:Video.AVI";

gCapParms.fMakeUserHitOKToCapture= FALSE;
gCapParms.fCaptureAudio= TRUE;
gCapParms.wPercentDropForError= 100;
gCapParms.wNumVideoRequested =                   gCapParms.fUsingDOSMemory ? 32 :
1000;

// If wChunkGranularity is zero, the granularity will be set to the
// disk sector size.
gCapParms.wChunkGranularity = (gbIsScrncap ? 32 : 0);
capCaptureSetSetup(ghWndCap, &gCapParms, sizeof(CAPTUREPARMS));

// set a filename for the captured video
// hWnd == Application Main Window Handle
capFileSetCaptureFile(hwnd, vidName);
gCapParms.wNumVideoRequested = 10;
gCapParms.wNumAudioRequested = 5;
gCapParms.fLimitEnabled = TRUE;
if (gCapParms.wTimeLimit == 0)
    gCapParms.wTimeLimit = 5;

// Inform the capture window of the capture settings
capCaptureSetSetup(ghWndCap, &gCapParms, sizeof(CAPTUREPARMS));

// Capture video sequence to file specified by cmdSetCaptureFile()
fResult = capCaptureSequence(ghWndCap);
```

2.4.2 Video Overlay

2.4.2.1 Overview

The overlay engine provides a method of merging video capture data with the graphics data on the screen. Supported data formats include YUV 4:2:2, RGB15, RGB16, and RGB24. The source data can be mirrored horizontally or vertically or both. Overlay data comes from a buffer located in local memory or AGP memory. Data can be double buffered using a pair of 32-bit buffer pointer registers. Data can either be transferred into the overlay buffer from the host or the video capture logic. Buffer swaps can be done by the host and internally synchronized with the display VSYNC. Buffer swaps also automatically happen based on the completed capture frame from the video capture engine and the display VSYNC.

The Intel740™ graphics accelerator overlay can accept line widths up to 1024 pixels. Each image can be enlarged using a 6-bit expansion value filtered in both the horizontal and vertical directions. The horizontal filter is a 3-Tap FIR type and the vertical filter uses either line replication, smoothing at line boundaries, or continuous running average.

2.4.2.2 Field Based Content

When interlaced video data is stored in progressive field format, the field-based method (also referred to as the “bob” method) of displaying video data is used to show each field individually using an overlay. Vertical scaling is required to stretch the image to the original aspect ratio on a progressive PC monitor. Each field is half the normal height. Thus, for example, it can be vertically zoomed by 2X using an overlay stretch to restore the correct aspect ratio.

Due to the spatial interlacing of the top and bottom fields, proper vertical position adjustment is required to align the two fields. To prevent the image from jittering up and down, the initial phase of the overlay vertical scalar is set to one for the top field. This accommodates the one source line offset between the two fields. This method produces a 60 fields per second (NTSC) field display on progressive monitors and retains all temporal information.

The bob field-based display method can be automatic when showing video from the capture engine. The Intel740™ graphics accelerator is capable of using field information from incoming video to automatically adjust the overlay vertical position (auto-bob method).

2.4.3 VBI and Intercast

2.4.3.1 Overview

A vertical blanking interval (VBI) is the time period in which a television signal is not visible on the screen because of the vertical retrace (that is, the electron gun repositioning to the top of the screen to start a new scan). Data services can be transmitted using a portion of this signal. In a standard NTSC signal, roughly 10 scan lines are potentially available per channel during the VBI. Each scan line represents a data transmission capacity of about 9600 baud.

When special data is mixed with video data, as it is for VBI or Intercast, the Intel740™ graphics chip’s scalars should not be used. It is important to use a capture chip which can send scaled video data with raw VBI data. The Intel740 graphics chip will accept the VBI data and video data into the same capture buffer where software can separate the two forms of data.

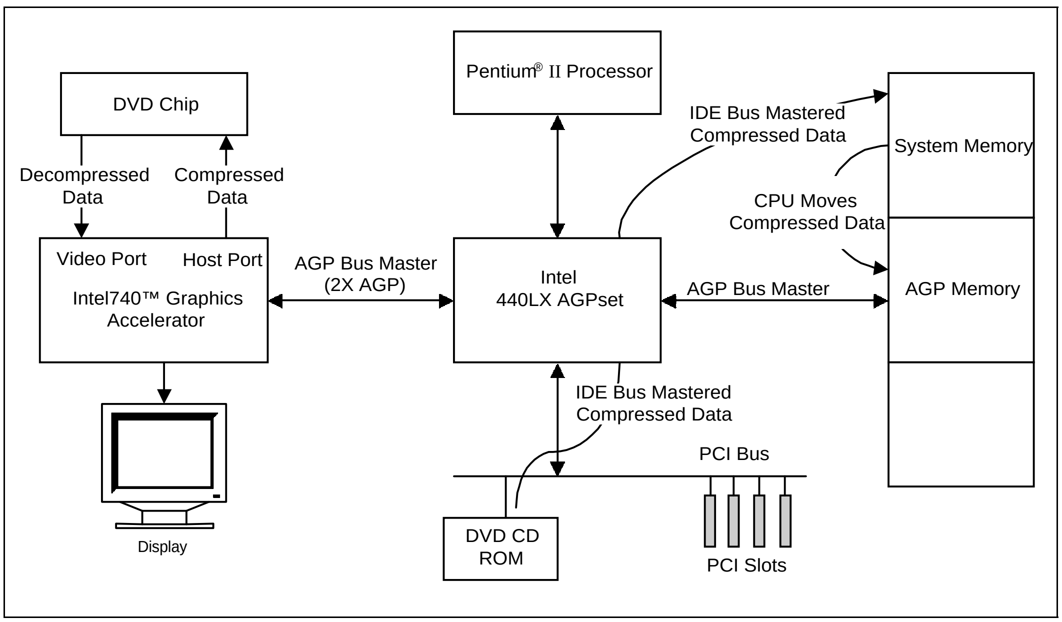
2.5 DVD Capabilities

2.5.1 Overview

The Intel740 chip's VMI port consists of a video port and a host port. The host port provides an enhanced VMI 1.4 Mode B port; the enhancements allow burst modes of operation. The Intel740 chip's video port is used to receive decompressed video data from a DVD chip, a video decoder chip, or from a software decoder. Using both the host and video ports, DVD, TV, Intercast, and video capture can be achieved. The incoming video stream sent to the Intel740 chip is in YUV2 format with a resolution of 720x480 at 30 frames per second following the CCIR601 8-bit pixel standard. Use of the Intel740 chip's overlay capability allows images from the capture engine to be displayed while being captured. Figure 2-23 illustrates the flow of data through a system performing DVD playback. Typically, a DVD drive will be attached as an EIDE device where the DVD compressed data is bus mastered into main memory. The data flow steps are:

1. IDE bus master DVD data to main memory
2. CPU moves compressed data to AGP memory
3. Intel740 chip uses AGP bus master to move compressed data to the DVD decoder chip (via the host port)
4. Decompressed data for the display is then sent back to the Intel740 chip through the video port

Figure 2-23. Data Flow for DVD Playback



2.5.2 Hardware DVD/MPEG-2 Movie Playback

2.5.2.1 Software Considerations

The Intel740 chip drivers, DirectDraw HAL and DDVPE HAL handle video display for DVD. The VMI interface is handled by the Intel740 chip drivers, while the video port is supported through a VPE interface. Using the Intel740 chip drivers for the host port VMI and VPE for the video port, there is no need to write directly to the Intel740 chip's video configuration registers or VMI.

2.5.2.2 Creating a VPE Port

Appendix A provides the vpe.h and vpp.cpp files as an example of how to create a VPE port. This is sample code only and is intended to be used as a help for those unfamiliar with VPE. This code takes input from the 8-bit VMI video port and displays it using VPE.

2.6 TV Out Interface

2.6.1 Overview

The Intel740 graphics accelerator chip has a digital TV out interface. When using this interface, normal VGA display cannot be used. The 12-bit digital interface is designed to interface with an external TV encoder, which incorporates a high quality flicker filter and performs overscan compensation. While the Intel740 chip supports the TV out interface, not all specific card implementations will support this feature. Refer to the individual graphics card documentation to see if TV out is supported.

When TV out is enabled, the pixel data from the Intel740 chip must be supplied at a rate required by the TV out port. This means that the PC monitor must run at about 60 Hz refresh when NTSC TV out is desired, and 50 Hz when PAL is desired.

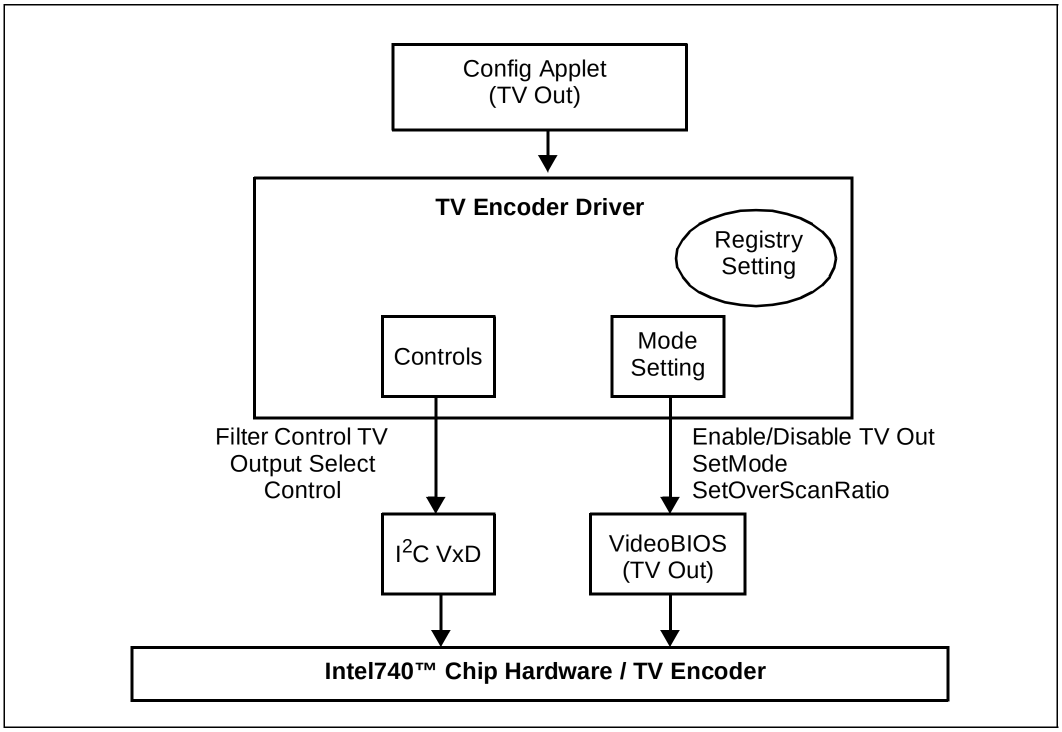
Only the following screen modes need to be supported for TV out:

- 720x400 Text (like VESA Mode 3+ for TV out support during boot-up)
- 640x480
- 800x600
- 320x240 (for TV-Out DOS game support)
- 320x200 (for TV-Out DOS game support)
- 720x480 (Windows 9x only, Maximum Overscan, and no Flicker-Filter)
- 720x576 (Windows 9x only, Maximum Overscan, and no Flicker-Filter)

All modes should support the same color depths that are supported by the Intel740 chip without TV out enabled, since the TV out hardware is not concerned with how the Intel740 chip generates the pixel data at it’s TV out port.

On the software side, other than for special encryption cases, TV out requires no special programming. The Windows* control panel handles all functions of the TV out capabilities (the software structure is diagrammed below in Figure 2-24). The only programming required for TV out is during playback of digital video (typically DVD) that calls for copy protection. This is discussed further below.

Figure 2-24. Windows* TV Output Control Software Structure



2.6.2 Using TV Out with Copy Protection

One requirement of DVD playback is that the TV signal be encrypted so that an external recording device (a VCR) cannot copy the signal. Microsoft* has introduced a new standard way of enabling this in Microsoft Windows 98*, using the VIDEOPARAMETERS escape described in the Win98 Driver Design Kit. The Intel740 chip's TV out driver also provides a custom interface for operating systems (Win95) that do not support the Windows 98 standard. This is through a call to the SetMovieMode interface. Both methods are shown below.

2.6.2.1 Enabling Copy Protection Using SetMovieMode

SetMovieMode

Syntax:

```
HRESULT SetMovieMode (WORD wCopyProtectMode)
```

Description:

Sets the copy protection mode to use while encoding. If the TV Out encoder must support copy protection, it is enabled with this function. Only the application that sets his mode may disable it, by calling SetMovieMode(0).

Parameters:

```
WORD wCopyProtectMode
```

The copy protection mode to use. If this value is 0, copy protection is disabled.

Return:

```
S_OK
```

The copy protection mode change occurred successfully,

```
E_FAIL
```

The encoder hardware does not support copy protection.

```
E_ACCESSDENIED
```

Copy Protection could not be disabled. Another application may have set the movie mode. Only the application that sets protection may disable it.

Example 2-37. Disabling Copy Protection Using SetMovieMode

```
#include "gfxTVOut.h"
...
CoInitialize();
...
ITVOut* pTVOut;
HRESULT hr = CoCreateInstance(CLSID_TVOut, NULL, CLSCTX_INPROC_SERVER,
    IID_ITVOut, (void**)&pTVOut);
if (SUCCEEDED(hr))
{
    if (SetMovieMode(0) == S_OK)
    {
        MessageBox(NULL, "Copy protection disabled", "Notice", MB_OK);
    }
    else
    {
        MessageBox(NULL, "Unable to set copy protection.", "Notice",MB_OK);
    }
}
else
{
    MessageBox(NULL, "Problem Creating TVOut interface","Error",MB_OK);
}
CoUnnitialize();
```

2.6.2.2 Enabling Copy Protection Using VIDEOPARAMETERS (Win98)

This is the recommended way of enabling copy protection under Windows 98*. SetMovieMode will only work for the Intel740 graphics accelerator driver, whereas the method described here should work for any Windows 98 TV out driver.

The Windows 98 API for TV capabilities uses a structure called VIDEOPARAMETERS to interface with the video card. The structure definition is given below.

```
typedef struct _VIDEOPARAMETERS {
GUID Guid;
DWORD dwOffset;
DWORD dwCommand;
DWORD dwFlags;
DWORD dwMode;
DWORD dwTVStandard;
DWORD dwAvailableModes;
DWORD dwAvailableTVStandard;
DWORD dwFlickerFilter;
DWORD dwOverScanX;
DWORD dwOVerScanY;
DWORD dwMaxUnscaledX;
DWORD dwMaxUnscaledY;
DWORD dwPositionX;
DWORD dwPositionY;
DWORD dwBrightness;
DWORD dwContrast;
DWORD dwCPType;
DWORD dwCPCommand;
DWORD dwCPStandard;
DWORD dwCPKey;
BYTEbCP_APSTriggerBits;
BYTEbOEMCopyProtection[256];
} VIDEOPARAMETERS, *PVIDEOPARAMETERS, FAR *LPVIDEOPARAMETERS;
```

To effectively change the current copy protection setting for the TV out function, the dwCommand parameter must be set to VP_COMMAND_SET, and the dwCPCommand parameter to VP_CP_CMD_ACTIVATE. The change is activated by calling ChangeDisplaySettingsEx (see Example 1-4). Once copy protection has been enabled, the dwCPKey variable will contain a copy protection key value. This value will need to be set in order to deactivate copy protection by setting dwCPCommand to VP_CP_CMD_DEACTIVATE.

More information regarding the VIDEOPARAMETERS structure and TV out settings can be referenced from the Microsoft Windows 98* Driver Design Kit.

Example 2-38. Enabling Copy Protection using the VIDEOPARAMETERS Structure

```
#include "tvout.h"
DEVMODE dm;
VIDEOPARAMETERSvp;

dm.dmSize = sizeof(DEVMODE);
vp.Guid = vpguid;
vp.dwCommand = VP_COMMAND_GET;

// Get current TV settings
if (ChangeDisplaySettingsEx("\\\\.\\Display1", &dm, 0,
CDS_VIDEOPARAMETERS, &vp) == DISP_CHANGE_SUCCESSFUL)
```

```
{
vp.dwCommand = VP_COMMAND_SET;
vp.dwCPCommand = VP_CP_CMD_ACTIVATE;

if (ChangeDisplaySettingsEx("\\\\.\\Display1", &dm, 0,
           CDS_VIDEOPARAMETERS, &vp) == DISP_CHANGE_SUCCESSFUL)
        {
           MessageBox(NULL, "Copy protection enabled",
"Notice", MB_OK);
        }
        else
        {
           MessageBox(NULL, "Error in Setting Copy Protection",
"Notice", MB_OK);
```

2.7 2X AGP Interface

For Intel740™ graphics accelerator accesses to the graphics aperture (located in system memory), the AGP interface to the host bridge is used. The interface is AGP 1.0 compliant. Bus operations are permitted in both 1X and 2X mode. Full 2X AGP implementation is integrated into the Intel740™ graphics accelerator with sideband operations supporting Type 1, Type 2, and Type 3 sideband cycles. Type 3 support permits textures to be located anywhere in the 32-bit system memory address space.

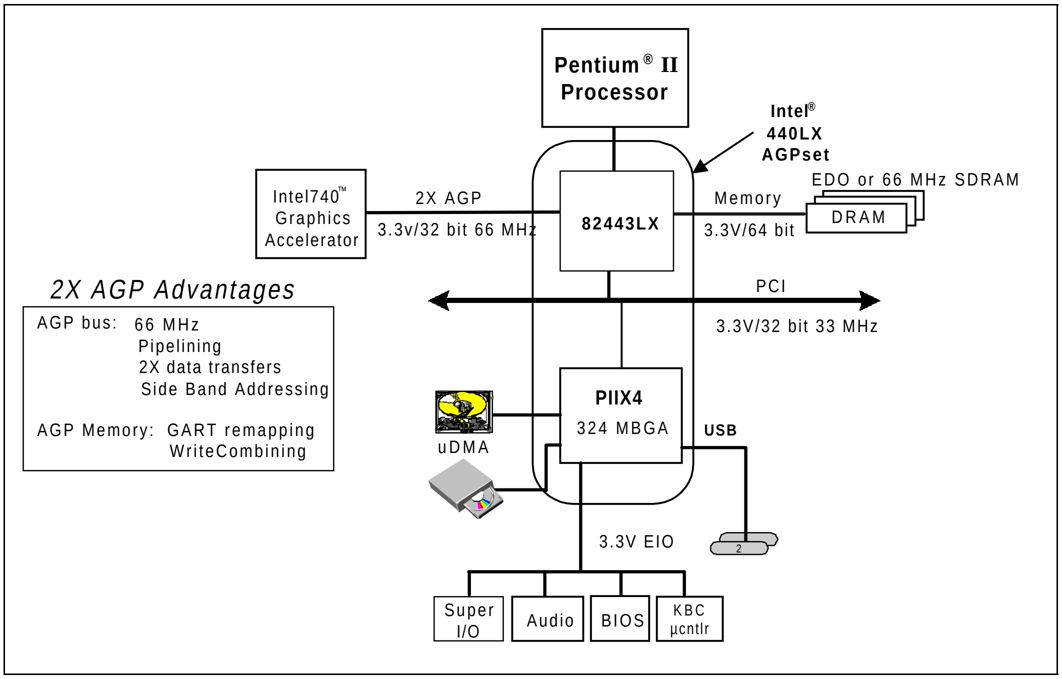
Combined with side-band addressing, the Intel740™ graphics accelerator is capable of achieving the highest AGP performance possible. The side-band addressing allows the Intel740™ graphics accelerator to issue requests without having to wait for data to be written or returned from the host. Internal buffering within the Intel740™ graphics accelerator accounts for any latency over AGP. This buffering allows the various internal pipelines to proceed at full processing speed without having to wait for data.

Using 2X AGP, the various memory surfaces can be stored and executed directly from AGP memory (DME). Being executed directly from AGP memory allows an Intel740™ graphics accelerator system to store large textures efficiently for very realistic 3D rendering. When executing directly from AGP memory, the Intel740™ graphics accelerator orders its accesses to minimize page breaks and maximize memory efficiency.

2.7.1 AGP Primer

The Accelerated Graphics Port (AGP) brings new levels of performance and realism to next-generation 3D graphics accelerators. The principal benefit comes from the graphics accelerator having high speed access to surface textures and other graphics surfaces in main system memory. Special performance oriented AGP features allow much faster read/write access to these surfaces than has been possible in the past. The basic memory architecture of an AGP system is illustrated in Figure 2-25.

Figure 2-25. Intel740™ Graphics Accelerator Connects to System Memory Over AGP



Graphics software infrastructure requires that AGP memory be contiguous, which means a page based system memory must have a graphics address remapping table (GART) capability. This is because the operating system ordinarily allocates randomly located pages of memory whereas graphics software requires its memory to be contiguous.

The translation facility gives each memory page a second aliased address. All the addresses are adjacent, making this part of system memory closely resemble conventional video memory. Memory accessible through the GART is referred to as non-local video memory, meaning video memory that is not local to the Intel740™ graphics accelerator.

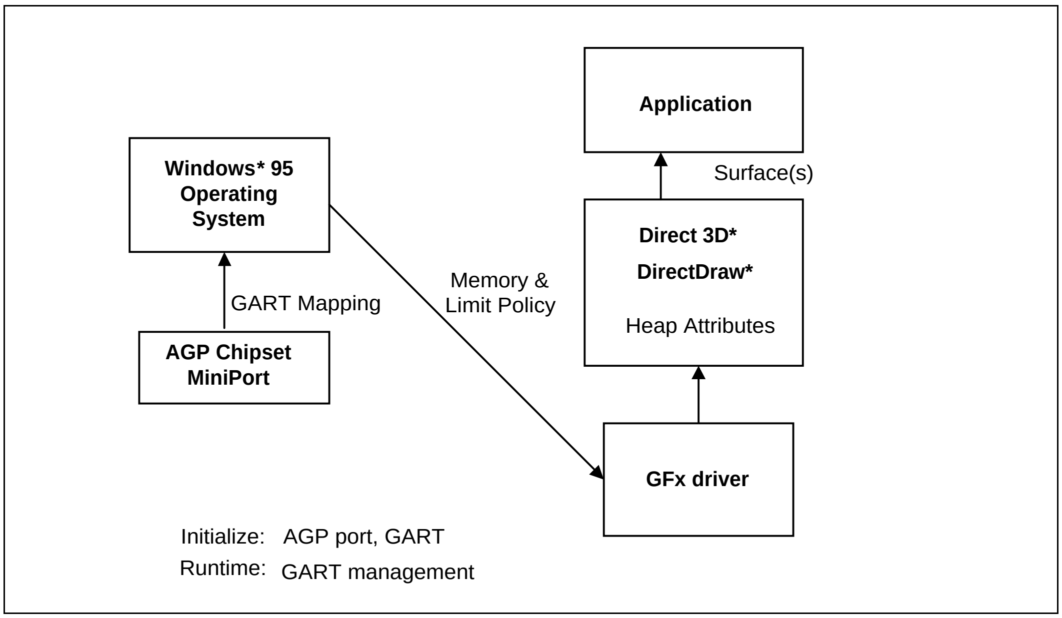
Non-local memory can be accessed by the host processor, by the Intel740™ graphics accelerator, and in current AGP systems by other PCI devices. In future systems, the GART translation will only be used by AGP graphics devices and the host processor will perform a corresponding address translation.

2.7.2 AGP Software Architecture

DirectDraw applications request space for graphics surfaces by calling the DirectDraw function “CreateSurface.” Space for the surface is obtained from heaps defined by the graphics driver. Memory for non-local memory heaps is obtained from the operating system. When more non-local video memory is needed, DirectDraw can obtain additional memory from the operating system. Memory is locked in place and mapped into the proper GART address range. The surface is aligned and its memory type established as specified in the graphics heap template. Requests to expand AGP memory are honored so long as the total amount of AGP memory does not exceed a limit set by the operating system.

Initialization details are attended to at the time the operating system is loaded. The operating system calls the chipset miniport which initializes AGP port parameters, allocates space for the GART translation table, initializes the GART hardware, and performs the actual AGP memory allocation/deallocation. The interaction of these functions is summarized in Figure 2-26.

Figure 2-26. New Services in Windows Work with DirectDraw to Support AGP Applications



2.8 BIOS Interface

The Intel740™ graphics accelerator supports a maximum video BIOS size of 256K x 8. Flash can be used.

2.9 Local Memory

The Intel740™ graphics accelerator uses SDRAM technology and can interface to SGRAM through its 64-bit memory interface. Memory Bus speeds range from 66 MHz to100 MHz while configurations of 2, 4 and 8 Mbytes are supported. Using a 64-bit interface, up to 800 Mbytes/s peak bandwidth is supported. The Intel740™ graphics accelerator allows operands to be placed in either local video memory or AGP memory. It is recommended that the Z, display, and Render buffers, video capture and MPeg overlay be located in local video memory, however when space becomes limited, the Render buffer should be relocated into AGP memory.